ViNO: SDN Overlay to Allow Seamless Migration Across Heterogeneous Infrastructure

Spandan Bemby, Hongbin Lu, Khashayar Hossein Zadeh, Hadi Bannazadeh, and Alberto Leon-Garcia
Dept. of Electrical and Computer Engineering,
University of Toronto
{spandan.bemby, hongbin.lu, k.hosseinzadeh, hadi.bannazadeh, alberto.leongarcia}@utoronto.ca

***Abstract*—We propose ViNO (Virtual Network Overlay), an orchestration service that can be used to create arbitrary network topologies with OVS (Open vSwitch) switches and VMs. ViNO connects switches and VMs through an overlay network using VXLAN encapsulation. ViNO provisions VMs by making API calls to the underlying platform. Users specify the desired topology using an expressive Domain Specific Language that allows users to easily express commonly used network topologies while hiding the underlying complexity. An important use case for ViNO is enabling the seamless migration of Linux services across VMs in different regions with very little downtime.**

Orchestration, SDN, Cloud computing

I. Introduction

Future application platforms will be based on the principles of cloud computing and software-defined networking (SDN) [1]. Cloud computing leverages the virtualization of physical computing resources to simplify management and to allow sharing of resources [2]. SDN separates a network's control plane from its data plane to provide both fine control and flexibility to perform complex tasks.

Such environments pose new challenges that involve computing and networking, specifically: 1) the placement and migration of VMs and their interconnecting network, as demand or capacity changes, or in response to failures; 2) the transition period where applications must operate over SDN-enabled islands and legacy networks. We present an approach for creating overlay networks that leverage SDN capabilities to address these challenges. **ViNO** (**Vi**rtual **N**etwork **O**verlay) is an orchestration service that creates arbitrary network topologies with Open vSwitch (OVS) switches (OVS is a production quality virtual multilayer switch)[6] and VMs. We have developed ViNO in the context of a major Canadian project to design and deploy a testbed to explore future application platforms within the NSERC Strategic Network for Smart Applications on Virtual Infrastructures (SAVI) [3]. The SAVI Testbed (TB), operates with OpenStack and OpenFlow enabling it to exploit the increasing capabilities of these two initiatives.

II. Overview

Consider experiments that require a network configuration that may not be directly realizable. For instance, an experiment may require more VMs-than can be actually placed- to be placed within a specific broadcast domain. In general, there may be experiments where computational and network requirements conflict. This problem can be overcome using overlay networks created through tunneling protocols like VXLAN.

Now consider a user who may be interested in prototyping a network topology. The experiment may consist of creating VMs, configuring them, retiring them, dynamically creating and connecting new VMs, etc. This is typically done either manually, or through ad-hoc scripts. Doing this manually can be prohibitively time consuming because the user needs to: 1) know the VM provisioning APIs of many cloud platforms, 2) know how to configure switches, and 3) remotely log into multiple VMs to cross configure them.

To solve this problem, we propose a tool that orchestrates the creation of arbitrary network topologies- ViNO. ViNO is an orchestration tool for the SAVI Testbed that can manage the lifecycle of an experiment. This includes the creation, retirement, and modification of nodes (a term to collectively refer to VMs and soft-switches) and their interconnections. To use ViNO, the user specifies the topology using a domain specific language (DSL). The DSL is parsed by ViNO, which realizes the infrastructure using the API of the underlying VM provisioning module.

In this paper we also show results of live migrating Linux services (containers) across different VMs. A container is a form of lightweight virtualization that allows multiple isolated instances of the system to be run on a single operating system [4]. The containers retain their state, including running processes, and their MAC and IP addresses, respectively. The key advantage of Linux containers is that they are platform agnostic [4]. This allows us to use and migrate containers across heterogeneous infrastructures, including different clouds like the SAVI TB, Amazon EC2, and legacy computing servers.

Migration also demonstrates the need for SDN. Without SDN, the migration of MAC and IP addresses would cause a complicated reconfiguration of the network that would disrupt network service. These disruptions prevent the migration from being seamless since the migrated services becomes unavailable for long periods, i.e. until a traditional network learns the new location of MAC and IP addresses. With SDN, we can proactively install flow rules, which can make network forwarding reconfiguration and the migration, seamless.

We have 2 primary contributions: 1) the design and implementation of ViNO and a DSL for easily describing arbitrary topologies, 2) the seamless migration of services across het-

```
1 nodes['sw1'] = {'contr_addr': '10.12.11.26:
  6633', 'region':'CORE', 'flavor':
  'm1.small', 'bridge_name': 'sw1_br', 'int_ip':
  ('p1', '192.168.200.18')}
  nodes['sw2'] = {'contr_addr': '10.12.11.26:
2 6633', 'region':'EDGE-TR1', 'flavor': 'm1.
  small'}
3 nodes['h1'] = {'region':'CORE', 'flavor': 'm1.
  tiny'}
4 nodes['h2'] = {'region':'CORE', 'flavor': 'm1.
  tiny'}
5 topology['sw1'] = [('h1', '192.168.200.10',
  'h1_br' )]
6 topology['sw2'] = ['sw1', ('h2',
  '192.168.200.11')]
```

Fig. 1. An example of a topology declaration using the DSL.

erogeneous infrastructure.

III. Design

A. Architecture

ViNO allows the orchestration of networks that consist of VMs, switches, and their interconnections. The switches are softswitches, specifically OVS switches. ViNO handles provisioning VMs and configuring some of these VMs as switches. The overlay interconnection between VMs is created through VXLAN encapsulation[5]. VXLAN can create arbitrary Layer 2 (L2) or Layer 3 (L3) networks by encapsulating L2 frames in L3 UDP packets.

To use ViNO, users need to authenticate themselves by specifying their credentials in the *config* file. Next, the users specify the topology using the DSL, in the *topology* file. Once the topology is specified, the user can instantiate the topology using the *ViNOLauncher* script, which provisions the VMs, configures them, and configures the overlay network. ViNO parses the topology and provisions the VMs on the SAVI Testbed. If the user intends to modify the topology at run-time, they should use the *SetupNodes* and *SetupTopology* scripts to create nodes, and setup the interconnections, respectively.

B. Python Based DSL

The user specifies a topology and the interconnections between hosts and switches using a Python-based DSL. The DSL is meant to be declarative and expressive. The language supports multiple regions.

Figure 1 shows the declaration of an example topology. Lines 1 and 2 show the declaration for a switch. Lines 3 and 4 show the declaration for VMs. Lines 5 and 6 shows how a switch is connected to a host, and the overlay IP address of the host. This is the IP address that is associated with the VXLAN interface that connects the switch to the host. Note, this IP address is different from the IP address that the VM provisioning system assigns to the VM.

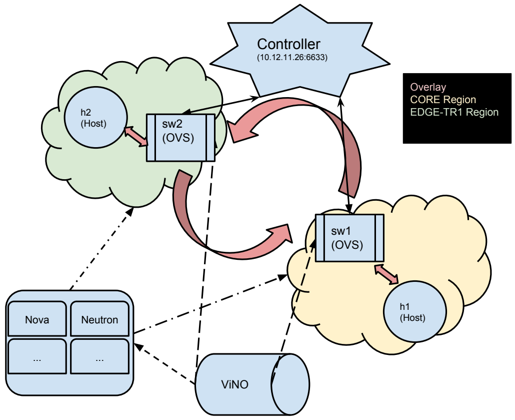

Fig. 2. Example topology and the connection between elements. A dashed arrow represents a procedure call or a configuring event. A solid arrow represents a connection between elements. ViNO makes calls to Nova and Neutron. Nova and Neutron are components of the cloud management platform used by SAVI TB and OpenStack. Other cloud services like Amazon, have a similar architecture.

C. Setting Up Nodes

OpenStack exposes a RESTful API to provision VMs. The API allows us to launch VMs and configure various parameters, such as the image to be used. ViNO parses the topology file and determines how many VMs to launch. It then determines which of these to configure as switches and which to configure as hosts. ViNO determines the value for each configuration parameter for each node by reading the specified value or using the default if a value is unspecified. The *SetupNodes* script uses the nova client to send requests to the server to provision the VMs.

D. Setting Up Topology

The nodes are connected using logical links created using VXLAN tunnels. The VXLAN header contains a 24 bit VXLAN Network Identifier (VNI) field. VMs are only allowed to communicate with VMs with the same VNI [5]. VXLAN overlay segments exist between VXLAN Tunnel End Points (VTEP). The VTEPs are responsible for encapsulating and decapsulating VXLAN headers. An OVS bridge acts as a VTEP. To configure the VMs and switches, ViNO creates a bridge inside the VM (the bridge is needed to send and receive packets over the VXLAN overlay). Next, it creates another port for VXLAN communication. Then, ViNO sets the internal IP address of this port. The newly created port can be configured and assigned an IP address. Figure 3 illustrates how the VXLAN overlay works.

E. Dynamically Modifying Topology and Adding Nodes

To dynamically change the topology, the user must change the topology file and run the two scripts, *SetupNodes* and *SetupTopology*, to create nodes and setup the overlay networks, respectively.

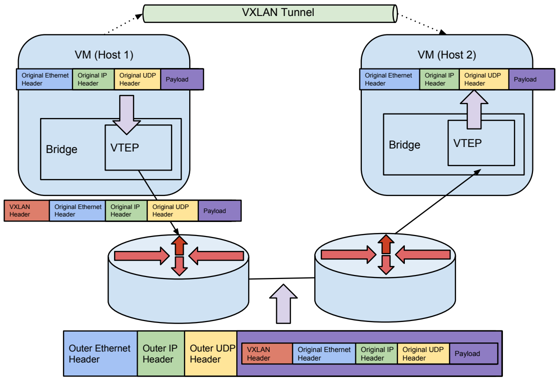

Fig. 3. An illustration of how VXLAN works.

F. Migration

The SAVI Testbed consists of two types of clouds: Core (remote datacenters) and Smart Edge (local datacenters). This architecture allows application administrators to optimize the performance of their applications by placing their VMs depending on the latency-bandwidth requirements of the applications. It may be desirable to change the placement of VMs at runtime, in response to changing workloads.

VM migration [10], [11] allows a service to be dynamically relocated. When a VM is migrated, the states of the CPU, memory, and disk are transferred. However, cross-datacenter migrations, due to separate management domains and heterogeneity of the underlying infrastructure, do not allow us to maintain the IP address of the VMs.

One solution to this is to use nested virtualization [12], [13]. In this scheme, there are two layers of virtualization: layer 1 consists of hypervisors running on physical machines that host VMs; layer 2 consists of layer 1 VMs emulating hardware to further host VMs. In this case, the management plane resides in the user space, and so IP addresses can be maintained across migrations. However, there are some shortcomings of this approach. VMs are typically stored in vendor specific formats, such as VMWare's vmdk, Oracle Virtualbox's vdi, and Amazon EC2's ami, which poses portability issues when moving VMs across environments [8]. An even bigger concern is performance- due to: 1) the large size of a VMs, 2) layer 2 VMs being emulated (rather than virtualized). Initially, we considered this scheme using QEMU [7]. We found it challenging to migrate the hosted VMs across heterogeneous platforms and migration took a prohibitively large amount of time and had a high resource overhead.

An alternative is to have the layer 2 virtualization consist of containers. Primitives required to support containers are supported by most distributions. This makes Linux containers platform independent and allows heterogeneous deployments. Compared with VMs, containers share the kernel with other containers and therefore have a lower overhead and can be migrated faster [8]. For our experiment we used OpenVZ containers, which is a mature containerization project that supports live migration [9]. Using this scheme we migrated containers across datacenters in the SAVI Testbed. We built layer 2 virtualization by performing the following steps:

1) Provision VMs on SAVI TB; install OpenVZ on the VM.
2) Create and configure OpenVZ containers.
3) Construct an overlay private network to connect all OpenVZ containers.

The overlay network constructed in step 3 is an independent network that uses a separated IP address space, which was constructed in the following manner.

1) Configure OpenVZ to ensure that each newly created OpenVZ container will automatically obtain a private IP address from the same subnet.
2) Connect collocated OpenVZ containers by an OVS bridge (we had to modify the OVS kernel module to be able to compile it with the OVZ kernel).
3) Use ViNO to setup overlay topology to connect containers.

It is possible to migrate OpenVZ containers across different datacenters without changing its IP address. This prevents interruptions due to migration, for end-users who are connected to the hosted application.

Although we envisioned an overlay system that enables a seamless application relocation, deploying such a system is a non-trivial process because it involves provisioning VMs from different datacenters, installing the necessary middleware on the provisioned VMs and cross-configuring the VMs to setup the overlay network topology. For a large system that requires a large number of VMs and a sophisticated network topology, the deployment process will be infeasible. Furthermore, deploying such systems requires knowledge from many different domains (e.g. networking, operating system, cloud computing)- which system deployers may not have. To address these challenges, we created ViNO to automate the deployment process. ViNO allows us to specify complex network topologies and the overlay network connecting VMs in a human-readable language. This is crucial in facilitating the deployment process and reducing human errors. Figure 5 illustrates how VM migration works.

IV. Evaluation

We conducted two experiments to evaluate ViNO. First, we evaluated the performance of ViNO in setting up different network topologies; second, we demonstrated the feasibility of a system that supports migration of Linux container by using ViNO. In both experiments, ViNO was hosted by a server with 4 virtual CPUs and 8GB of RAM.

A. Experiment 1

The goal was to evaluate the performance of ViNO in setting up various network topologies on the SAVI TB. Each VM had 1 virtual CPU and 2GB of RAM, using the CentOS image with OVS pre-installed. It took roughly 30 seconds for ViNO to setup a 3, 6, 9, and 12 node topologies, and it took roughly 50 seconds to setup a 24-node topology. We

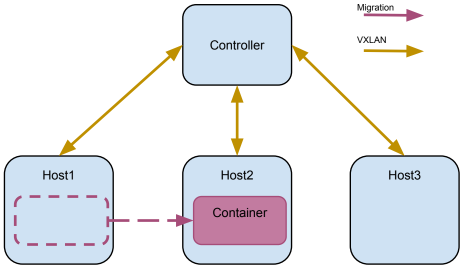

Fig. 4. The deployment overlay network.

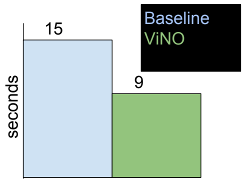

Fig. 5. Total migration time.

observed that the time to setup a topology was independent of the number of nodes since ViNO automatically parallelizes the tasks executing on each node. However, for large topologies, we observed moderate performance degradation.

B. Experiment 2

We used ViNO to construct an overlay network and evaluated the performance of migrating a Linux container across datacenters. The setup included two VMs provisioned across 2 data centers Initially, there was a container on host 1 running a Tomcat HTTP server. We simulated a client by using JMeter[1], which would send HTTP requests every 0.5 seconds. We then migrated the the Tomcat container to host2, and measured the response time.

We measured *application downtime*, the time that the application is unable to serve its clients. Downtime is composed of : 1) migration time, and 2) network refractory time.

The experiment was conducted under two settings. First, the switches were learning switches. Second, we used ViNO to deploy OpenFlow switches that were connected to a Ryu[2] controller, which removed the switches learning time. Figure 8 shows that we were able to reduce the application downtime from 15 to 9 seconds by using ViNO and SDN.

V. Related Work

Mininet [14] is a network emulator that creates virtual hosts, links, and switches all on a single Linux kernel. This makes Mininet unusable when one wants to test an application with a large number of interconnected entities, with realistic delays.

MaxiNet [15] is a SDN network emulator that extends Mininet across several physical machines. MaxiNet employes a traffic generator, to emulate realistic traffic conditions. MaxiNet is an emulation framework that sits on top of physical machines; whereas ViNO sits on top of the OpenStack platform. This makes extending ViNO easier.

Distributed OpenFlow Testbed (DOT) [16] is an emulator for SDN. DOT emulates a network topology across a cluster of physical machines. The physical machines emulate the hosts and the virtual switches. Virtual machines can be located on different physical machines, and interconnected with GRE tunnels.

VI. Conclusion

ViNO orchestrates overlay networks, and allows users to do things that would have been otherwise infeasible. ViNO has many use cases. In particular, the ability to live migrate VM containers across heterogeneous platforms with minimal downtime is very powerful and has significant implications for datacenters.

Acknowledgment

This research was supported by the National Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] Converged Infrastructure. http://www.cioandleader.com/cioleaders /features/7505/converged-infrastructure
[2] Amazon. What is Cloud Computing. http://aws.amazon.com/what-is-cloud-computing/
[3] SAVI Network. http://www.savinetwork.ca/
[4] Linux Containers. https://linuxcontainers.org/
[5] VXLAN Draft. http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-00
[6] Open vSwitch: B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, Extending Networking into the Virtualization Layer. In Proceedings of Hotnets, 2009.
[7] Qemu. www.qemu.org
[8] Docker. https://github.com/docker/docker
[9] OpenVZ. http://openvz.org/
[10] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt and A. Warfield, ”Live migration of virtual machines,” in Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, Boston, MA, 2005, pp. 273-286.
[11] R. Bradford, E. Kotsovinos, A. Feldmann and H. Schiberg, ”Live wide-area migration of virtual machines including local persistent state,” in Proceedings of the 3rd international conference on Virtual execution environments, San Diego, CA, 2007, pp. 169-179.
[12] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har’El, A. Gordon, A. Liguori, O. Wasserman and B.-A. Yassour, ”The turtles project: design and implementation of nested virtualization,” in Proceedings of the 9th USENIX conference on Operating systems design and implementation, Vancouver, BC, 2010, pp. 1-6.
[13] D. Williams, H. Jamjoom and H. Weatherspoon, ”The Xen-Blanket: virtualize once, run everywhere,” in Proceedings of the 7th ACM European conference on Computer Systems, Bern, 2012, pp. 113-126.
[14] Mininet: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#what
[15] P. Wette, M. Draxler, A. Schwabe, F. Wallaschek, M.H. Zahraee and H. Karl, MaxiNet: Distributed Emulation of Software-Defined Networks in Proceedings of IFIP, Trondheim, 2014, pp. 1-9.
[16] A.R. Roy, M.F. Bari, M.F. Zhani, R. Ahmed, R. Boutaba, ”Design and management of DOT: A Distributed OpenFlow Testbed” in Proceedings of NOMS, Krakow, 2014, pp. 1-9.

[1] http://jmeter.apache.org/
[2] https://github.com/savi-dev/ryu